Random Graphs for Performance Evaluation of Recommender Systems

Szymon Chojnacki and Mieczysław Kłopotek

Institute of Computer Science,
Polish Academy of Sciences
J.K. Ordona 21, 01-237 Warsaw, Poland
{sch, klopotek}@ipipan.waw.pl

Abstract. The purpose of this article is to introduce a new analytical framework dedicated to measuring performance of recommender systems. The standard approach is to assess the quality of a system by means of accuracy related statistics. However, the specificity of the environments in which recommender systems are deployed requires to pay much attention to speed and memory requirements of the algorithms. Unfortunately, it is implausible to assess accurately the complexity of various algorithms with formal tools. This can be attributed to the fact that such analyses are usually based on an assumption of dense representation of underlying data structures. Whereas, in real life the algorithms operate on sparse data and are implemented with collections dedicated for them. Therefore, we propose to measure the complexity of recommender systems with artificial datasets that posses real-life properties. We utilize recently developed bipartite graph generator to evaluate how state-of-the-art recommender systems' behavior is determined and diversified by topological properties of the generated datasets.

Keywords: recommender systems, performance evaluation, random graphs, bipartite complex networks

1 Introduction

Recommender systems are an important component of the Intelligent Web. The systems make information retrieval easier and push users from typing queries towards clicking at suggested links. We experience real-life recommender systems when browsing for books, movies, news or music. The engines are an essential part of such websites as *Amazon*, *MovieLens* or *Last.fm*. Recommender systems are used to deal with the tasks that are typical for statistical classification methods. They fit especially the scenarios in which the number of attributes, classes or missing values is large. Classic data-mining techniques like *logistic regression* or *decision trees* are well suited to predict which category of news is the most interesting for a particular customer. Recommender systems are used to output more fine-grained results and point at concrete stories.

In recent years we have observed a surge of interest of research community in recommender systems. One of the events that was responsible for this phenomenon was the Netflix Prize challenge [1]. The competition was organized by a large DVD retailer in US. The prize of 1 million dollars was awarded to the team that managed to improve RMSE (root mean standard error) of the retailer's Cinematch algorithm by more than 10%. It turned out that classic collaborative filtering techniques [2] do not perform as good as SVD-based (Singular Value Decomposition) approaches [3]. During the competiotion a new method derived from the field of artificial neural networks[1] was applied with high accuracy (i.e. Restricted Boltzmann Machines [5]). The most successful solution was achieved by blending various algorithms [6].

The lesson we learned during the Netflix Prize is that the difference between the quality of simple methods and the sophisticated ones is not as significant as we could have expected. Moreover, in order to lower RMSE an ensemble of complex and computationally intensive methods has to be done. Even though the organizers made much effort to deliver realistic and huge data, the setting did not envision the problems that we need to face in diverse real-life recommender systems applications, such as:

- the *Cold Start* problem, i.e. an arrival of new users with short history (e.g. restricted to the last HTTP session)
- instant creation of new items (e.g. news, auction items or photos)
- real-time feedback from users about our performance

These drawbacks were overcome during the Online Task of the Discovery Challenge [7] organized as a part of the ECML 2009 (European Conference on Machine Learning). The owners of the `www.BibSonomy.org` bookmarking portal opened its interfaces to recommender systems taking part in the evaluation. Whenever a user of BibSonomy was bookmarking a digital resource (a publication or a website) a query was sent to all the systems. The tag recommendation of a random one was displayed to the user. After the action a feedback with user's actions was sent to all systems. The systems could have been maintained during the challenge, because they were configured as web services. The results showed that all of the teams found it difficult to deliver majority of its recommendations within time constraint of 1 000 ms.

Our research was motivated by the above result and an observation that the development of recommender systems is limited by a fact that there are not enough possibilities to test the algorithms with various datasets. The data structure used by recommender systems is a sparse $user \times item$ matrix with ratings. It is a hard exercise to generate randomly such matrices with predefined properties resembling real-life situations, because of three reasons. Firstly, if we fix the number of users, items and rankings and try to place the rankings randomly in the matrix we obtain symmetric distribution of the number of items

[1] Recently new algorithm [4] based on the concept of *k-separability* was developed with promising results, which shows that the power of artificial neural networks in this domain has not been fully harnessed yet.

rated by users (and vice versa). However, in real-life datasets the distributions are skewed. Secondly, simple random selection results in no correlations among user's preferences. Such correlations exist in real datasets. Thirdly, if we generate one matrix with some desired properties and would like to add new users or items, we would probably loose the properties of the original matrix.

We have challenged the problem recently [8]. We proposed to look at the matrix with ratings as if it was a bipartite graph with nodes of both modalities representing users and items respectively. A rating from the matrix is mapped onto an edge in the bigraph. We proposed an algorithm in which we can control not only simple statistics like numbers of users, items or rankings, but also obtain skewed distributions and correlations among users or items. Moreover, our random bigraph generator's asymptotic properties were verified by virtue of formal and numerical tools and we can add user or items to the graph without loosing the properties of the original datasets. The algorithm was obtained by adapting the advances in unipartite complex networks modeling onto a bipartite ground. We modified the preferential attachment model of Barabási and Albert [9] with the extension of Liu [10] and generalized the *surfing mechanizm* by Vázquez [11].

In this paper we apply the generator to produce several random bigraphs with various properties and evaluate how the properties impinge on the performance of analyzed recommender systems. We analyze four features of the systems that in our opinion are responsible for the success of an algorithm in a real-life setting:

- time required to build a model from a scratch
- memory consumption of the trained model
- latency of creating a recommendation
- time of updating the model with new ratings

We considered six algorithms during the tests: UserBased, ItemBased, SlopeOne, UserThreshold, KnnItem and SVD [12]. We relied on high-performance Mahout library [13]. Our attention was focused on five properties of artificial datasets: (1) size of the graph, (2) relative number of edges, (3) proportion of the number of users to the number of items, (4) clustering of edges, (5) distributions of node degrees of both modalities. We also checked the influence on the performance UserBased model of two characteristics: similarity measure and the size of the neighborhood.

Throughout the article we show that our approach can be used to better understand the features of datasets that are responsible for the performance of recommender systems. We utilize this knowledge to show (1) which algorithms are best suited for various scenarios, (2) identify datasets' features that are correlated with improving performance of some algorithms and diminishing performance of others, (3) point at potential directions of improving the implementations of the algorithms.

The rest of the article is organized as follows. In Section 2 we describe how recommender systems are evaluated. In Section 3 we outline the details of applied random bigraph generator. The fourth section contains the results of extensive experiments. The last fifth section is dedicated for the concluding remarks.

2 Performance of Recommender Systems

In our research we perceive performance in terms of real-life speed of using an algorithm. We omit analysis of statistical indicators such as accuracy, recall, f-measure, RMSE or lift charts. This is because, even though they are crucial in the process of selecting an algorithm to be deployed, it seems pointless to evaluate these measures within randomly generated datasets. One could argue that the usefulness of artificial datasets is questionable. And the best strategy is to evaluate all possible algorithms with one's real dataset and choose the most accurate that gives recommendations within specified time constraints. We argue with this point of view. Based on our experience we are more likely to believe that the structure and topology of datasets changes rapidly and the performance may be affected by appearance of outlying observations or unexpected growth in scale. It is hard to foresee all potential pitfalls and the need to evaluate the algorithms within a wide range of artificial data emerges. In order to justify our deduction we present in Fig. 1 the results of an online evaluation of the systems that participated in the social bookmarking Discovery Challenge organized as a part of ECML'09.

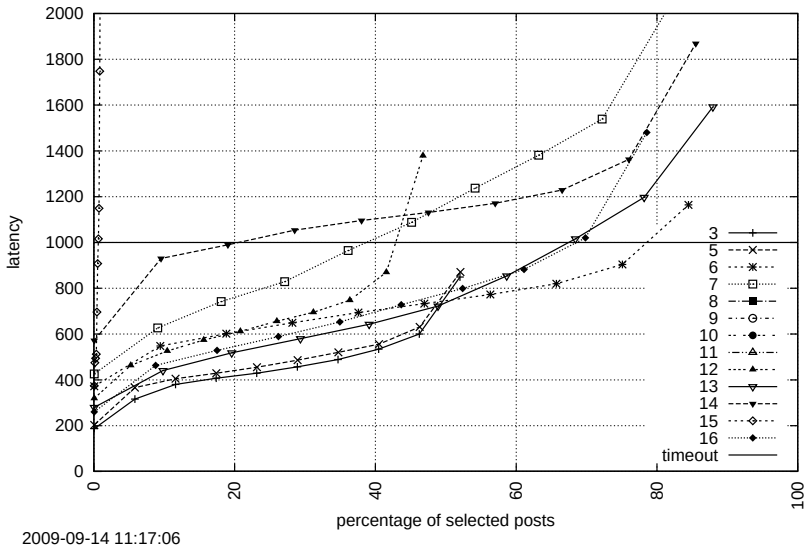

Fig. 1. Latency of recommender systems evaluated during the Online Task of ECML-PKDD 2009 Discovery Challenge. None of the systems delivered recommendations to all queries. The most recommendations were output by system number 13 (over 80%). The points on curves show the percentiles of latency for each algorithm. For example the second most right point point of algorithm 13 indicates that almost 80% of its recommendations were delivered within 1 200ms. Source: http://www.kde.cs.uni-kassel.de/ws/dc09/results/online/.

The algorithms were deployed as web services and latency was measured as the difference of time between sending a request and receiving a list of top five recommended tags. We participated in the evaluation and managed to lower the latency of our system to the level of 400 ms [14]. It turned out to be an important improvement for the users of the BibSonomy and resulted in the highest rate of clicks among all the evaluated systems.

A good starting point when analyzing complexity of any algorithm is to assess its asymptotic properties. Such analyses are usually based on several simplifying assumptions. One of the assumptions that is virtually never met is the dense data structure representation assumption. For example vectors are either implemented as dense (e.g. `double[]`) or sparse arrays (e.g. `ArrayList<Double>`). Dedicated collections are used to find optimal trade-offs between memory consumption and speed. Moreover, several advanced issues such as Java `Object` overhead and its influence on garbage collectors performance need to be taken into account. A great discussion about these problems is contained in chapter 3 of [12].

Nonetheless, theoretical analysis of complexity strengthens our general intuition about the upper bound of time needed for computations. A thorough analysis of wide range of recommender algorithms is described in [15]. In particular, it is assessed that training time, latency and memory consumption of SVD-based recommender are $O(E)$, $O(1)$, $O(U + I)$, where E is the number of ratings, U is the number of users and I is the number of items. The limitation of this kind of analysis is the fact that except of some rare situations it does not help us to decide which of two selected algorithms is expected to perform better on real data. An example of a set of soft rules trying to cope with such questions is drawn in Table 1.

Recommender	Key features
UserBased [2]	Fast when number of users is relatively small
ItemBased [16]	Fast when number of items is relatively small
SlopeOne [17]	Recommendations and updates fast at runtime
	Requires large precomputations
	Suitable when number of items is relatively small
KnnItem [18]	Good when number of items is relatively smaller
SVD [19]	Good results
	Requires large precomputations

Table 1. Comparison of recommender systems, based on [12].

In our experiments we did not find evidence to assert that UserBased algorithm performs significantly different than ItemBased when the proportion of the number of users to the number of items varies. We also found that there exist other factors than U, I or E that are interpretable and impinge on the performance in a coherent way. The results are discussed in detail in Sec. 4.

3 Bipartite Random Graph Generator

In this section we describe an algorithm used to generate random bigraphs. The algorithm was introduced and described in detail in [8]. In Sec. 3.1 we define the parameters of the algorithm, in Sec. 3.2 the properties of the generator are outlined.

3.1 Generative procedure

The generative procedure consists of three steps: (1) new node creation, (2) edge attachment type selection and (3) running bouncing mechanism. The steps are run after an initialization of the bigraph. The procedure requires specifying eight parameters:

m - the number of initial loose edges with a user and an item at the ends
T - the number of iterations
p - the probability that a new node is a user
$(1-p)$ is the probability that a new node is an item
u - the number of edges created by each new user
v - the number of edges created by each new item
α - the probability that a new user's edge is being connected to an item with preferential attachment
β - the probability that a new item's edge is being connected to a user with preferential attachment
b - the fraction of preferentially attached edges that were created via a *bouncing mechanism*

In the preferential attachment mechanism the probability that a node is drawn is linearly proportional to its degree. Opposite to the preferential attachment is random attachment, in which a probability of selection is equal for all nodes. The model is based on an iterative repetition of three steps.

Step 1 If a random number is greater then p create a new user with u loose edges, otherwise create a new item with v loose edges.

Step 2 For each edge decide whether to join it to a node of the second modality randomly or with preferential attachment. The probability of selection preferential attachment is α for new user and β for new item.

Step 3 For each edge that is supposed to be created with preferential attachment decide if it should also be generated via a bouncing mechanism.

Bouncing is performed in three micro steps: (1) a random node is drawn from the nodes that are already joined with the new node, (2) a random neighbor of the drawn node is chosen, (3) a random neighbor of the neighbor is selected for joining with the new node. The bouncing mechanism was injected into the model in order to parametrize the level of transitivity in a graph. The transitivity is a feature of real datasets and in terms of recommender systems represent the correlations between items ranked by different users. In unipartite graphs transitivity is measured by the *local clustering coefficient*, which is calculated for each node as a number of edges among direct neighbors of the node divided by all possible pairs of the neighbors. In bipartite graphs the coefficient is always zero

and is substituted by *bipartite local clustering coefficient (BLCC)* [8]. Bipartite local clustering coefficient of node j takes values of one minus the proportion of node's second neighbors to the potential number of the second neighbors of the node. The value of $BLCC$ calculated for node j is given by:

$$BLCC_j = 1 - \frac{|N_2(j)|}{\sum_{i \in N_1(j)} (k_i - 1)}, \tag{1}$$

where $|N_2(j)|$ stands for the number of the second neighbors of node j, $N_1(j)$ is a set of the first neighbors of node j and k_i is a degree of node i. The steps of the generator are depicted in Fig. 2

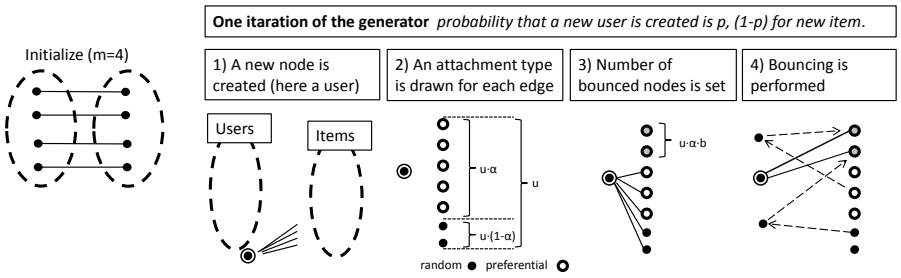

Fig. 2. For each edge of a new node, that is to be connected with an existing node with accordance to the preferential attachment mechanism, a decision is made whether to create it via a bouncing mechanism. In case of attaching new user node, u new edges are created. On average $u \cdot \alpha$ edges' endings are to be drawn preferentially and $u \cdot \alpha \cdot b$ of them are to be obtained via bouncing from the nodes that are already selected.

3.2 Properties

One can see that after t iterations the bigraph consists of $|U(t)| = m + pt$ users, $|I(t)| = m + (1-p)t$ items, and $|E(t)| = m + t(pu + (1-p)v)$ edges. Let's denote by η an average number of edges created during one iteration $\eta = (pu + (1-p)v)$. After relatively many iterations ($t >> m$) we can neglect m. In the presented model, an average user degree is:

$$\frac{|E(t)|}{|U(t)|} = \frac{m + t(pu + (1-p)v)}{m + pt} \approx \frac{\eta}{p},$$

Analogously an average item degrees is:

$$\frac{|E(t)|}{|I(t)|} \approx \frac{\eta}{(1-p)}.$$

The values are time invariant, but depend on both u and v. In Fig. 3, Fig. 4 and Fig. 5 three relations between model's parameters and graph's features are delineated.

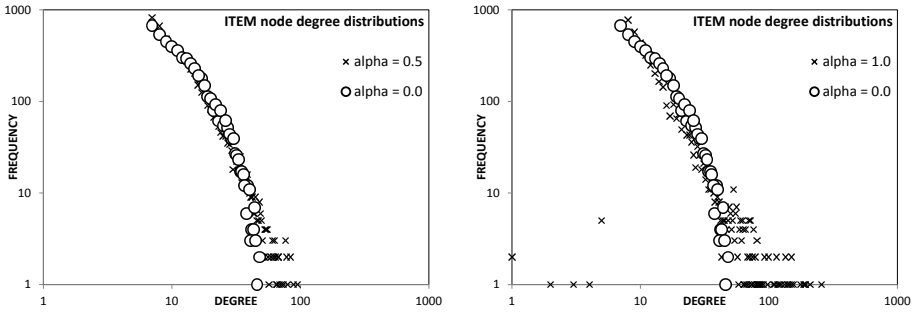

Fig. 3. Parameters α and β enable us to control the shape of node degree distributions. As the values approach unity, the shape becomes power-law, as the values vanish to zero the shape tends to exponential.

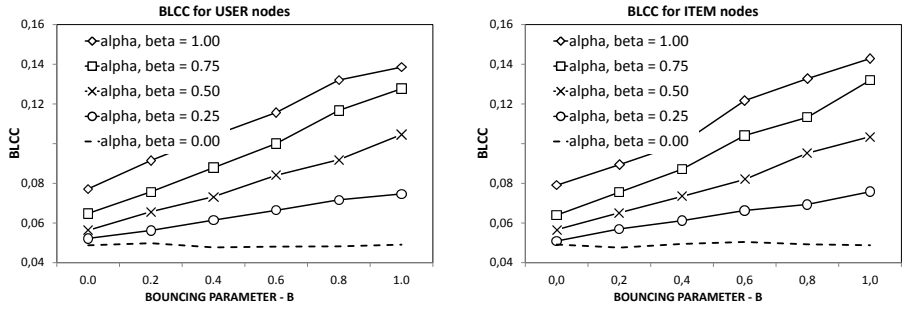

Fig. 4. The growth of parameter b results in higher values of BLCC. It does not influence BLCC if $\alpha = \beta = 0.0$.

4 Experiments

In order to evaluate the performance of analyzed algorithms we generated 83 artificial bipartite graphs. The statistics describing the graphs are contained in Fig. 14 and Fig. 15. Each graph's edge was augmented with a random integer from a set of possible rankings $\{0, 1, 2, 3, 4, 5\}$. After the last iteration (usually $T = 10\ 000$) hundred more edges were created by running 100 steps for each

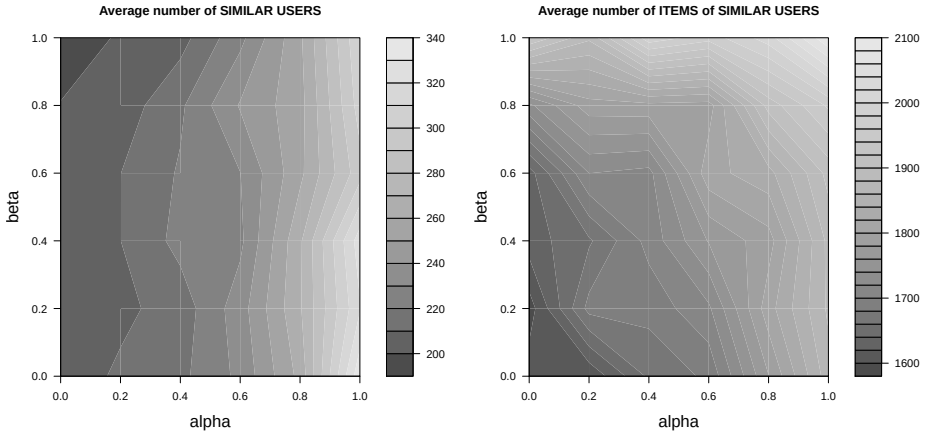

Fig. 5. The average number of similar users (having ranked at least one item in common with a defined user) and their items depends on both α and β.

graph with unchanged parameters. This enabled us to preserve asymptotic properties of the graphs within a set of rankings used to batch update of the models. The experiments were run in-memory within separate threads on a 64-bit Fedora operating system with the Quatro 2.66GHz Intel(R) Core(TM) i5 CPUs.

4.1 Evaluated systems

We evaluated six recommender algorithms implemented in the Mahout java library [13]. Mahout contains highly efficient open-source implementations of machine-learning algorithms maintained by a vibrant community. It is powering several portals e.g. *SpeedDate*, *Yahoo! Mail*, *AOL* or *Mippin*. The algorithms are: GenericUserBasedRecommender [2], GenericItemBasedRecommender [16], SlopeOneRecommender [17], GenericUserBasedRecommender with the neighborhood defined by non-negative threshold similarity [12], KnnItemBasedRecommender [18] and SVDRecommender [19]. The algorithms cover wide spectrum of approaches to the problems of *Collaborative Filtering*.

4.2 Performance Measures

In the subsequent subsections we study the correlations between the performance of the six algorithms differentiated by five graph properties: size, density, proportion of users to items, clustering and shape of node degree distributions. We focus our attention on four performance statistics:

1. **BUILD** - time in milliseconds that is required to load whole bigraph from a text file and train a model, after this period of time the model is ready to create recommendations

2. **MEMORY** - memory consumption in megabytes of the built model, it was assessed as a difference between `Runtime.getRuntime().totalMemory()` and `Runtime.getRuntime().freeMemory()` after calling a garbage collector five times
3. **LATENCY** - an average time in milliseconds required to produce a recommendation for a sample of 500 users
4. **UPDATE** - time in millisecond of updating a model with 100 new ratings

In case of UserBased recommender we set the size of the neighborhood to 200. SVD recommender was projected onto 10 factors and 200 iterations were run during training. The *Pearson Correlation* was used as a similarity measure. At the end we analyze specific parameters of UserBased model.

4.3 Scalability

In order to verify the ability of analyzed algorithms to scale we generated thirteen bigraphs. Each dataset was produced with the same parameters except of the number of iterations. The datasets are numbered from 10 to 22 in Fig. 14. The time of training SVD recommender may be misleading [2]. For most systems, only training time grows linearly with the size of a dataset. Memory consumption and latency grows sublinearly. SlopeOne and SVD algorithms exhibit the poorest performance in terms of building time and memory consumption. These costs pay back in the phase of creating recommendations. KnnItem does not scale during training and is the slowest during creating recommendations. Except of SlopeOne all models refresh their structures immediately. Only SVD's latency seems not to depend on the scale.

4.4 Density

We generated 14 bigraphs to test how performance depends on the density. The graphs are numbered from 23 to 36 in Fig. 14. All graphs have around the same number of nodes, but the number of edges varies between 30 100 and 240 100. The diversity was obtained by changing two parameters: u and v.

We omit the presentation of performance calculated for the first four bigraphs, in which $u = v$. All the four measures where growing steadily as we evaluated $(u = v = 3)$, $(u = v = 6)$, $(u = v = 12)$ and $(u = v = 24)$. Which could have suggested that there exists a strong correlation between the density and the performance. However, when we compare 5 pairs of graphs with $u \neq v$ (Fig. 7), the results become confusing. By setting $u = 3$ and iterating over $\{4, \ldots, 15\}$ with v the performance diminishes only slightly. If we freeze $v = 3$

[2] In the chart (Fig. 6) the time of SVD building was divided by 1 000 to get comparable results. Such long time of building the model is a result of running 200 iterations of the gradient descent. However, even a single iteration takes on average $1\,000/200 = 5$ times longer than building the other models.

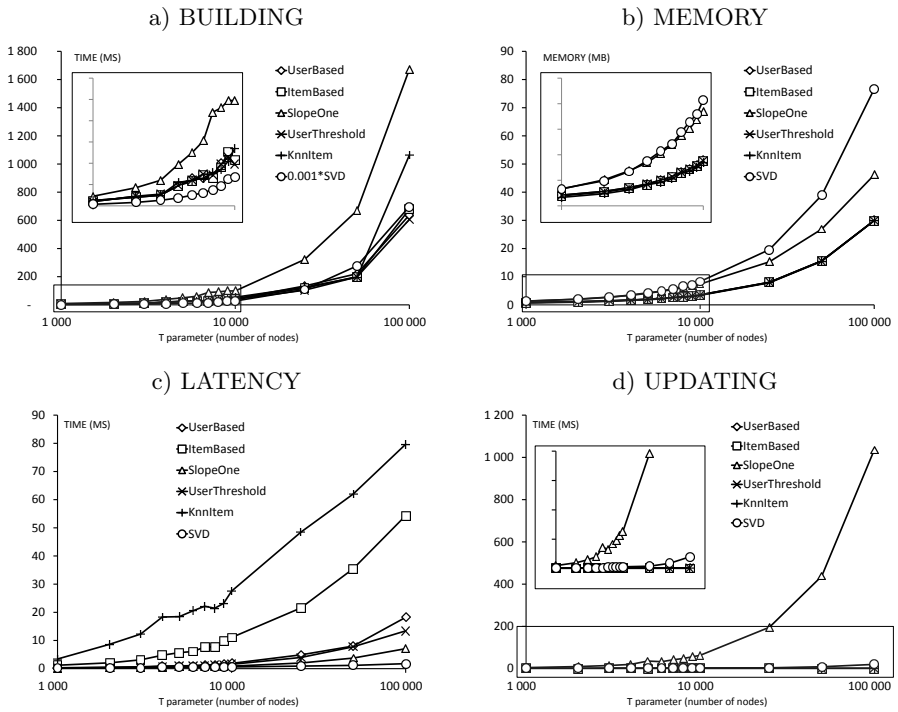

Fig. 6. Performance of the recommenders conditioned by the size of bigraphs. On all charts vertical axis has logarithmic scale. These enables to distribute evenly all observations.

and iterate with u we decrease the performance steadily. It suggests that the density on its own is not so much responsible for the performance[3].

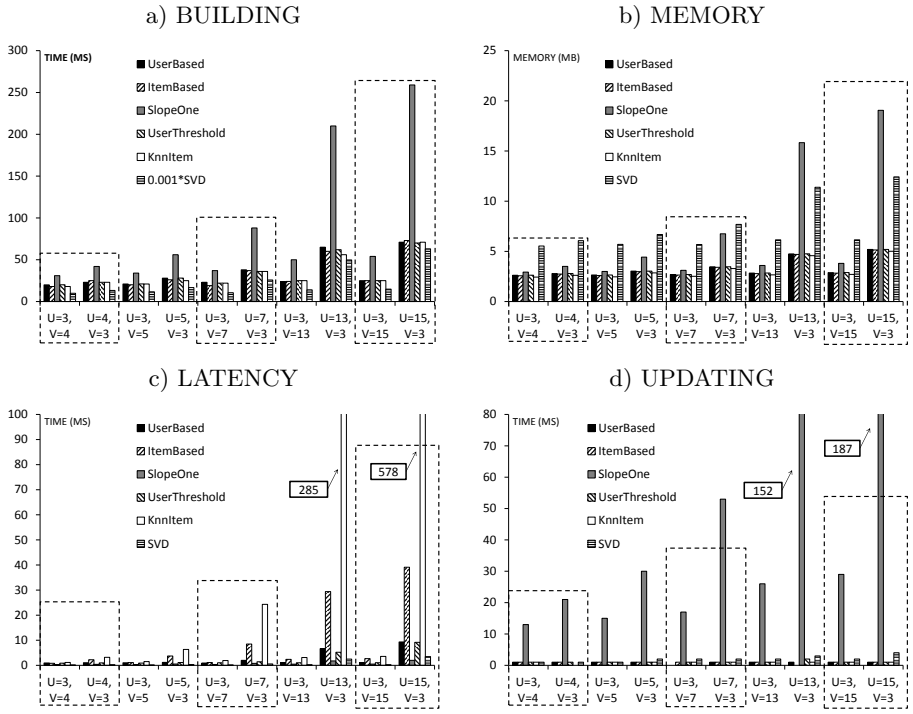

Fig. 7. Performance of the recommenders versus the density of the graphs.

4.5 Users to Items Proportion

Parameter p in the generator enables us to control the proportion of users to items. The parameter is interpreted as the probability that a new node is a user. We built nine graphs with constant numbers of nodes (= 10 200) and edges (= 70 100). The number of users varies between 1 166 and 9 082. The graphs are the first nine graphs in Fig. 14.

The relationship between p and the performance is depicted in Fig. 8. Only the performance of SVD does not seem to depend on the proportion in none

[3] Digging this effect deeper we see that by increasing v we not only increase the density, but also lower the variance of node degree distribution in user modality stronger than for the item modality. This shows that the performance relies heavily on the distributions of node degrees, but with different magnitude for both modalities.

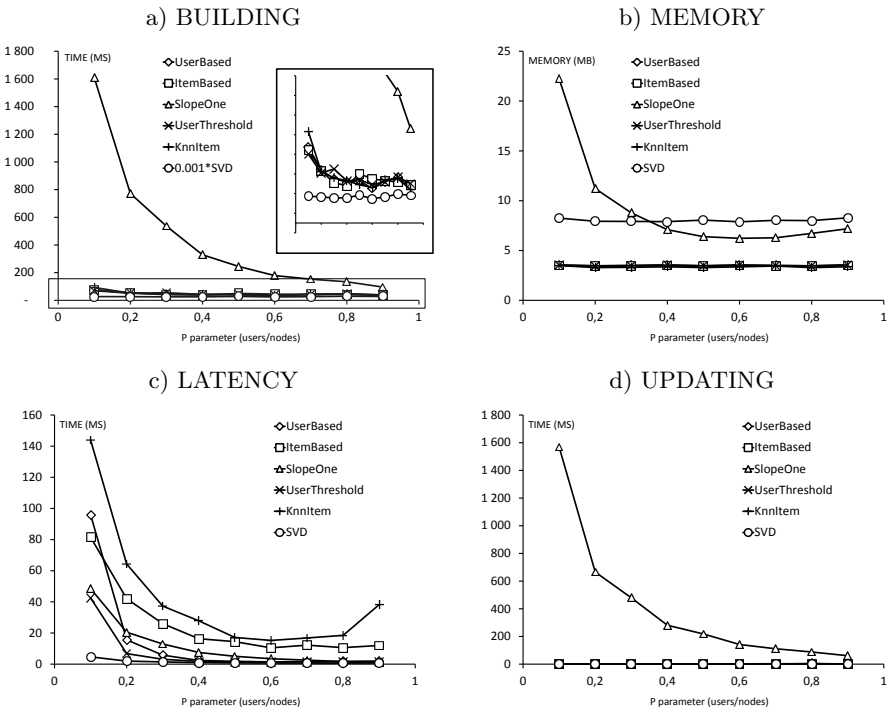

Fig. 8. Performance modeled by the proportion of users to all nodes.

of four evaluations. Training time decreases as the relative number of users increases for all remaining models. The only model, which memory consumption depends on the proportion is SlopeOne. It is very interesting that the lowest memory requirements are obtained for balanced graphs ($p \approx 0.5$). The same non-monotonic relation can be observed in case of KnnItem and latency. The rest of the algorithms improve their latency as the relative number of items grows. Even though most datasets have more items than users, the opposite situations can also happen. For example there are around $500K$ questions in `http://stackoverflow.com/` technical portal and only around $200K$ users. The fact that ItemBased recommender does not perform better than UserBased when there are relatively few items ([12]) raises our concern[4].

4.6 Clustering

Eleven graphs were generated to measure the influence of clustering on the performance. The datasets are numbered from 37 to 47 in Fig. 14 and Fig. 15. The bigraphs were generated by changing the bouncing parameter b from 0.0 to 1.0 with 0.1 intervals. All graphs have 1 100 nodes and 120 100 edges.

The performance of SVD does not depend on bouncing. Memory consumption and time of building grow slightly in line with clustering only in case of SlopeOne model. The latency of creating a recommendation is correlate with clustering gently only in case of ItemBase, UserBase, KnnItem and UserThreshold algorithms. The relation between the clustering and the performance is in all variants weak and not stable.

[4] This observation requires further investigation. In particular, we plan to verify if the claim that *UserBased algorithm are preferred over ItemBased when there are fewer users* is valid only when caching mechanisms are implemented.

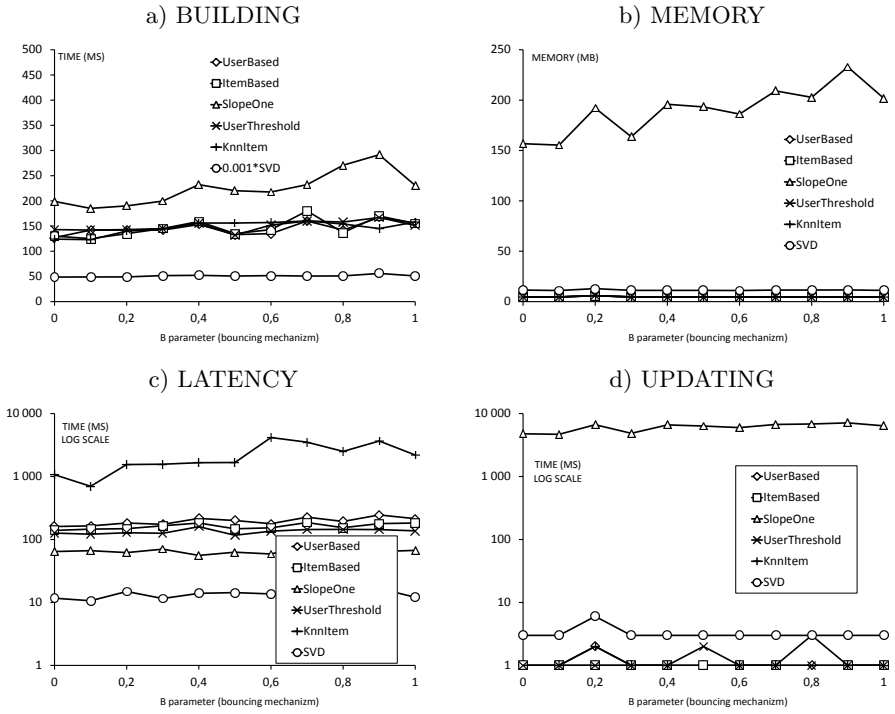

Fig. 9. Performance dimensioned by various levels of clustering (transitivity).

4.7 Shapes of degree distributions

We have shown in [8] that by changing α and β we can output graphs with a mixture of the power-law distribution and the exponential distribution (compare Fig. 3). The two parameters enable us to control independently the shapes of degree distributions of both modalities. By increasing α we can obtain graphs with constant number of users, items and edges, but growing average number of potentially similar users[5]. Moreover, an average number of distinct items of potentially similar users grows with both α and β (compare Fig. 5). These results are consistent with an asymptotic Newman's formula [20]. The formula is based on a local tree-like structure assumption and assess as an average number of second neighbors $|N_2(j)|$ of a random user j by means of the first moment of user degree distribution $\langle U \rangle$ and the first and the second moments of item degree distribution ($\langle I^2 \rangle$ and $\langle I^2 \rangle$ respectively):

$$|N_2(j)| = \langle U \rangle \left(\frac{\langle I^2 \rangle}{\langle I \rangle} - 1 \right). \tag{2}$$

We have generated 36 bigraphs with all possible combinations of α and β from a set of values $\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. All the graphs have 10 000 nodes and 70 100 edges. They are numbered from 48 to 83 in Fig. 15. We have observed that an average latency grows with either α or β for all analyzed algorithms (Fig. 11). We also observed that building time and memory consumption is related to α and β for all algorithms except for SVD and KnnItem. The performance of SlopeOne model is drawn in Fig. 10.

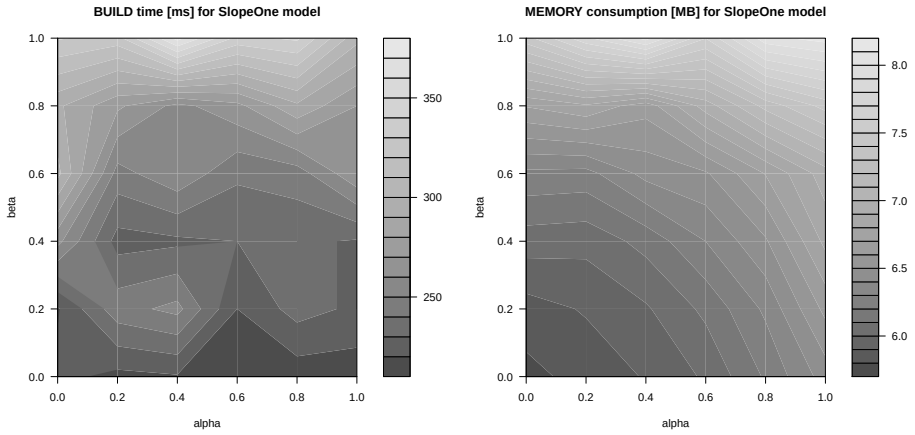

Fig. 10. Training time and memory requirements of SlopeOne model.

[5] We say that two users to be potentially similar if they have rated at least one item in common.

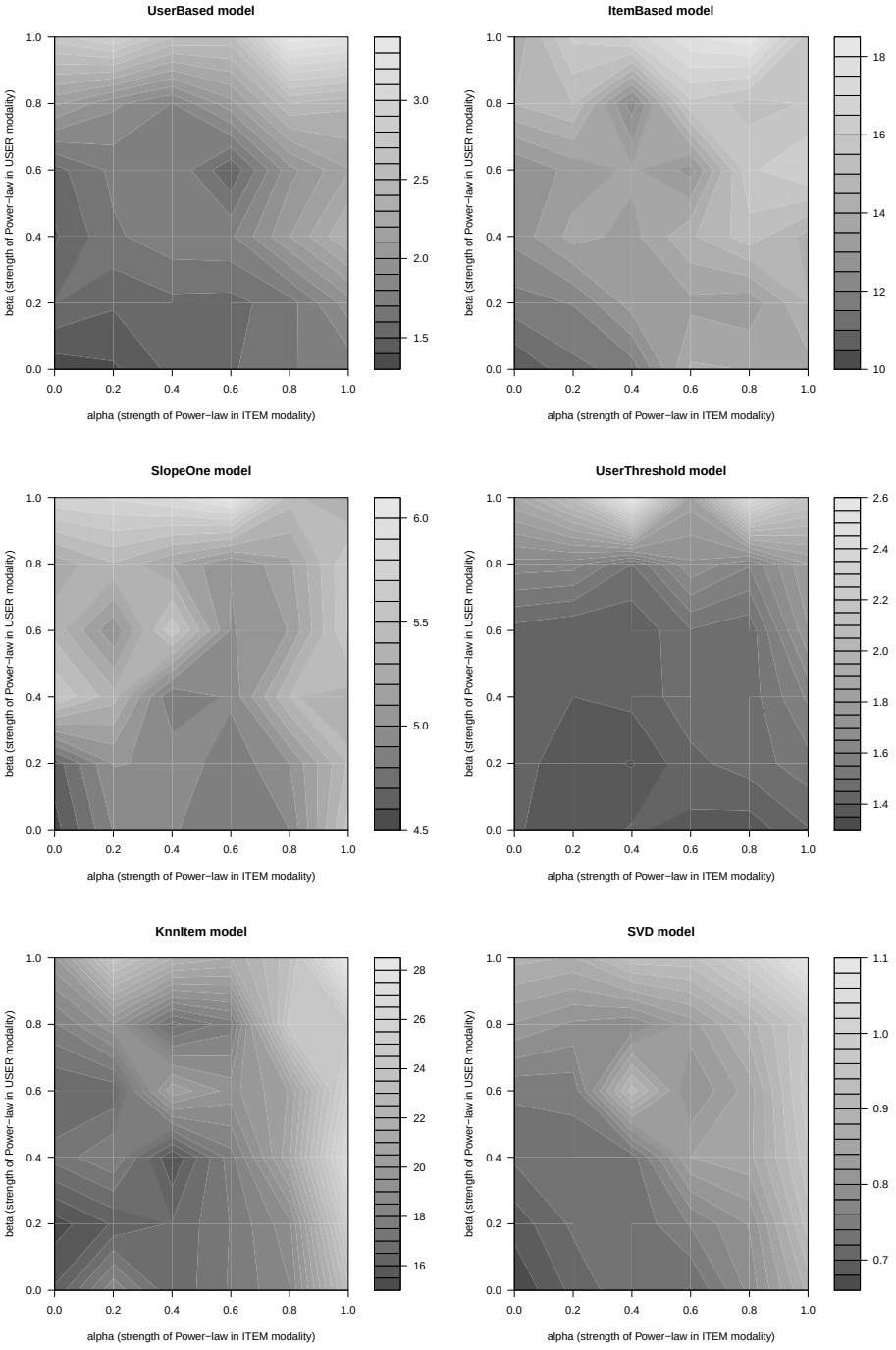

Fig. 11. The influence of α and β parameters on the latency.

4.8 Similarity measure

In the last two subsections we focus on UserBased algorithm and two parameters that are specific for it. The first parameter is the similarity measure and will be discussed in this subsection. The second parameter is the size of the neighborhood and will be described in next subsection.

The similarity between two vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ can be defined is several ways. We analyzed all that are available in the Mahout library i.e. *Pearson similarity*, *Euclidean similarity*, *LogLikelihood similarity*, *Spearmann similarity* and *Tanimoto similarity*. The third and the fifth are defined by set operations and can easily by used with *binary ratings*, when we only know if a user expressed a preference for an item or not. **Pearson similarity** is widely used and is related to the *Cosine similarity*. The only difference is that the former one operates on centered data. **Euclidean similarity** is negatively proportional to the euclidean distance between two vectors. **LogLikelihood similarity** is based on calculating four values: number of non-empty dimensions in both vectors, numbers of non-missing dimensions in the first and the second vector, number of all dimensions [21]. **Spearmann similarity** is calculated as Pearson similarity but x_i and y_j are substituted with their relative ranks i.e. the lowest value is 1, the second lowest is 2, and so on. **Tanimoto similarity** for binary data is calculated as a proportion of dimensions that are non-empty in both vectors by total number of non-empty dimensions in the vectors. Detailed definitions of all coefficients can be found in the javadoc API of Mahout [13].

We have observed in Fig. 12 that the latency of UserBased model depends on the selection of similarity measure. The fastest recommendations are output by Pearson and Euclidean similarities. Tanimoto and Loglikelihood similarities are around three times slower. Spearman similarity is the slowest and does not scale well with the growth of density.

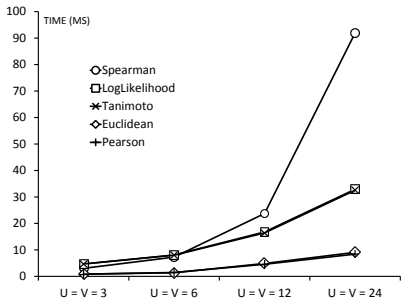

Fig. 12. Dependence of latency on similarity metrics. Four analyzed graphs have the same number of nodes, they are numbered $23 - 26$ in Fig. 14.

4.9 Size of the neighborhood

Size of the neighborhood is a parameter in UserBased recommender, which enables us to tune accuracy of the algorithm. When a UserBased model is requested to deliver recommendations for a given user two steps are performed. In the first step similarity of the user to all other users is calculated and only the most similar users are retained (the number is limited by the neighborhood). In the second step only items of the most similar users are weighted and the recommendation is selected among those items.

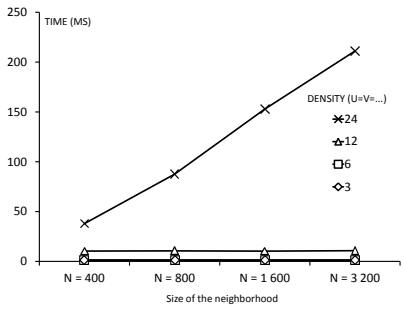

Fig. 13. Latency of creating a recommendation in four graphs with varying density. UserBased algorithm run with four different levels of the neighborhood.

The results in Fig. 13 indicate that as long as graph's density is below some threshold, the latency does not depend neither on the density nor size of the neighborhood. It can be explained by the fact that the neighborhood parameter is triggered only when the number of similar users is greater than it.

5 Conclusion

In the article we have proposed to use random bipartite graphs to measure the performance of recommender systems. We have showed that recently developed random bigraph generator [8] can be be used to generate a wide range of artificial datasets with predefined properties. The analytical framework can be used to compare various algorithms, but also to help us understand their complexity and point at potentially non-optimal implementations. We believe that the proposed methodology can be applied in various scenarios and settings. Further analyses need to be performed to understand the most intriguing results. In particular the real relationship between complexity of UserBased and ItemBased algorithms. And the emergence of U-shaped curve describing performance of various recommenders when the proportion of the number of users to the number of items is being changed. In the future we plan to evaluate how the performance of

recommender systems is altered when they are implemented in a distributed environment.

References

[1] "The netflix prize website." `http://www.netflixprize.com`.

[2] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 230–237, ACM Press, 1999.

[3] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proc. Int. Conf. on Knowledge Discovery and Data Mining*, pp. 426–434, 2008.

[4] G. Alexandridis, G. Siolas, and A. Stafylopatis, "An efficient collaborative recommender system based on k-separability," vol. 6354 of *LNCS*, pp. 198–207, 2010.

[5] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *Proceedings of the 24th International Conference on Machine Learning*, International Conference on Machine Learning, 2007.

[6] Y. Koren, "The bellkor solution to the netflix grand prize," 2009.

[7] R. Jäschke, F. Eisterlehner, A. Hotho, and G. Stumme, "Testing and evaluating tag recommenders in a live system," in *Workshop on Knowledge Discovery, Data Mining, and Machine Learning* (D. Benz and F. Janssen, eds.), pp. 44–51, 2009.

[8] S. Chojnacki and M. Klopotek, "Random graphs for bipartite networks modeling," *Journal of Control and Cybernetics*, 2011. to appear.

[9] A. Barabási and R. Albert, "Emergence of scaling in random networks," *Science (New York, N.Y.)*, vol. 286, no. 5439, pp. 509–512, 1999.

[10] Z. Liu, Y.-C. Lai, N. Ye, and P. Dasgupta, "Connectivity distribution and attack tolerance of general networks with both preferential and random attachments," *Physics Letters A*, vol. 303, no. 5-6, pp. 337 – 344, 2002.

[11] A. Vázquez, "Growing network with local rules: Preferential attachment, clustering hierarchy, and degree correlations," *Phys. Rev. E*, vol. 67, May 2003.

[12] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in action (MEAP)*. Manning, 2010.

[13] "The mahout website." `http://mahout.apache.org/`.

[14] S. Chojnacki, D. Czerski, and M. Kłopotek, "Optimization of tag recommender systems in a real life setting," in *3rd Conference on Human System Interaction*, pp. 107–112, 2010.

[15] M. Jahrer, A. Töscher, and R. Legenstein, "Combining predictions for accurate recommender systems," in *KDD '10*, pp. 693–702, ACM, 2010.

[16] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms.," in *WWW*, pp. 285–295, 2001.

[17] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," in *Proceedings of SIAM Data Mining (SDM'05)*, 2005.

[18] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights," in *ICDM '07*, pp. 43–52, IEEE Computer Society, October 2007.

[19] S. Zhang, W. Wang, J. Ford, F. Makedon, and J. Pearlman, "Using singular value decomposition approximation for collaborative filtering," pp. 257–264, 2005.

Lp.	Graph generator's parameters								Properties of generated graphs						
	m	T	p	u	i	alpha	beta	b	users	items	edges	neighbors	2nd neighbors	BLCC user	BLCC item
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(15)	(14)
1	100	10 000	**0,9**	7	7	0,5	0,5	0,0	9 082	1 118	70 100	962	977	4%	23%
2	100	10 000	**0,8**	7	7	0,5	0,5	0,0	8 105	2 095	70 100	547	1 456	3%	10%
3	100	10 000	**0,7**	7	7	0,5	0,5	0,0	7 060	3 140	70 100	379	1 795	3%	6%
4	100	10 000	**0,6**	7	7	0,5	0,5	0,0	6 187	4 013	70 100	295	2 049	3%	4%
5	100	10 000	**0,5**	7	7	0,5	0,5	0,0	4 998	5 202	70 100	233	2 524	3%	3%
6	100	10 000	**0,4**	7	7	0,5	0,5	0,0	4 157	6 043	70 100	211	3 064	4%	3%
7	100	10 000	**0,3**	7	7	0,5	0,5	0,0	3 081	7 119	70 100	203	4 142	6%	3%
8	100	10 000	**0,2**	7	7	0,5	0,5	0,0	2 172	8 028	70 100	219	5 511	10%	3%
9	100	10 000	**0,1**	7	7	0,5	0,5	0,0	1 166	9 034	70 100	249	7 809	22%	4%
10	100	**1 000**	0,9	7	7	0,1	0,1	0,0	993	207	7 100	253	202	11%	40%
11	100	**2 000**	0,9	7	7	0,1	0,1	0,0	1 897	303	14 100	405	296	9%	39%
12	100	**3 000**	0,9	7	7	0,1	0,1	0,0	2 833	367	21 100	528	357	8%	39%
13	100	**4 000**	0,9	7	7	0,1	0,1	0,0	3 716	484	28 100	611	460	7%	34%
14	100	**5 000**	0,9	7	7	0,1	0,1	0,0	4 600	600	35 100	620	560	6%	30%
15	100	**6 000**	0,9	7	7	0,1	0,1	0,0	5 558	642	42 100	683	600	5%	31%
16	100	**7 000**	0,9	7	7	0,1	0,1	0,0	6 420	780	49 100	686	710	4%	27%
17	100	**8 000**	0,9	7	7	0,1	0,1	0,0	7 291	909	56 100	699	807	4%	24%
18	100	**9 000**	0,9	7	7	0,1	0,1	0,0	8 235	965	63 100	742	851	4%	24%
19	100	**10 000**	0,9	7	7	0,1	0,1	0,0	9 120	1 080	70 100	753	936	3%	22%
20	100	**25 000**	0,9	7	7	0,1	0,1	0,0	22 627	2 573	175 100	920	1 791	2%	13%
21	100	**50 000**	0,9	7	7	0,1	0,1	0,0	45 214	4 986	350 100	986	2 690	1%	8%
22	100	**100 000**	0,9	7	7	0,1	0,1	0,0	90 192	10 008	700 100	1 023	3 730	1%	5%
23	100	10 000	0,9	**3**	**3**	0,1	0,1	0,0	9 063	1 137	30 100	141	264	0%	4%
24	100	10 000	0,9	**6**	**6**	0,1	0,1	0,0	9 106	1 094	60 100	554	843	2%	17%
25	100	10 000	0,9	**12**	**12**	0,1	0,1	0,0	9 126	1 074	120 100	2 051	1 069	11%	43%
26	100	10 000	0,9	**24**	**24**	0,1	0,1	0,0	9 113	1 087	240 100	5 652	1 087	37%	71%
27	100	10 000	0,9	**3**	**4**	0,1	0,1	0,0	9 110	1 090	31 090	147	287	0%	5%
28	100	10 000	0,9	**3**	**5**	0,1	0,1	0,0	9 076	1 124	32 148	157	318	0%	5%
29	100	10 000	0,9	**3**	**7**	0,1	0,1	0,0	9 084	1 116	34 164	164	364	1%	6%
30	100	10 000	0,9	**3**	**13**	0,1	0,1	0,0	9 076	1 124	40 340	198	523	1%	8%
31	100	10 000	0,9	**3**	**15**	0,1	0,1	0,0	9 089	1 111	42 232	215	575	1%	9%
32	100	10 000	0,9	**4**	**3**	0,1	0,1	0,0	9 108	1 092	39 108	252	469	1%	8%
33	100	10 000	0,9	**5**	**3**	0,1	0,1	0,0	9 133	1 067	48 166	386	641	2%	12%
34	100	10 000	0,9	**7**	**3**	0,1	0,1	0,0	9 123	1 077	66 192	728	866	3%	21%
35	100	10 000	0,9	**13**	**3**	0,1	0,1	0,0	9 077	1 123	119 870	2 121	1 092	12%	40%
36	100	10 000	0,9	**15**	**3**	0,1	0,1	0,0	9 114	1 086	138 268	2 721	1 070	15%	47%
37	100	10 000	0,2	12	12	0,9	0,9	**0,0**	2 039	8 161	120 100	419	8 118	29%	16%
38	100	10 000	0,2	12	12	0,9	0,9	**0,1**	2 128	8 072	120 100	414	8 025	29%	16%
39	100	10 000	0,2	12	12	0,9	0,9	**0,2**	2 026	8 174	120 100	405	8 135	30%	17%
40	100	10 000	0,2	12	12	0,9	0,9	**0,3**	2 146	8 054	120 100	410	8 004	30%	16%
41	100	10 000	0,2	12	12	0,9	0,9	**0,4**	2 065	8 135	120 100	402	8 093	31%	18%
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Fig. 14. Random graphs generated to test the performance of recommender systems (part 1). Graph generator's parameters and BLCC are defined in Sec 3.1. Neighbors stands for an average number of potentially similar users, 2^{nd} neighbors stands for an average number of items of potentially simialar users.

Lp.	Graph generator's parameters								Properties of generated graphs						
	m	T	p	u	i	alpha	beta	b	users	items	edges	neighbors	2nd neighbors	BLCC user	BLCC item
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(15)	(14)
42	100	10 000	0,2	12	12	0,9	0,9	**0,5**	2 106	8 094	120 100	397	8 046	32%	18%
43	100	10 000	0,2	12	12	0,9	0,9	**0,6**	2 100	8 100	120 100	396	8 051	32%	17%
44	100	10 000	0,2	12	12	0,9	0,9	**0,7**	2 022	8 178	120 100	393	8 130	33%	18%
45	100	10 000	0,2	12	12	0,9	0,9	**0,8**	2 013	8 187	120 100	384	8 144	34%	18%
46	100	10 000	0,2	12	12	0,9	0,9	**0,9**	2 031	8 169	120 100	378	8 117	35%	19%
47	100	10 000	0,2	12	12	0,9	0,9	**1,0**	2 120	8 080	120 100	381	8 025	34%	18%
48	100	10 000	0,5	7	7	**1,0**	**1,0**	0,0	5 070	5 130	70 100	327	3 273	4%	4%
49	100	10 000	0,5	7	7	**1,0**	**0,8**	0,0	5 058	5 142	70 100	329	3 097	4%	4%
50	100	10 000	0,5	7	7	**1,0**	**0,6**	0,0	5 054	5 146	70 100	325	2 943	4%	4%
51	100	10 000	0,5	7	7	**1,0**	**0,4**	0,0	5 130	5 070	70 100	372	2 955	4%	4%
52	100	10 000	0,5	7	7	**1,0**	**0,2**	0,0	5 027	5 173	70 100	322	2 803	4%	3%
53	100	10 000	0,5	7	7	**1,0**	**0,0**	0,0	5 125	5 075	70 100	330	2 702	3%	3%
54	100	10 000	0,5	7	7	**0,8**	**1,0**	0,0	5 141	5 059	70 100	266	3 049	4%	4%
55	100	10 000	0,5	7	7	**0,8**	**0,8**	0,0	5 114	5 086	70 100	270	2 856	4%	4%
56	100	10 000	0,5	7	7	**0,8**	**0,6**	0,0	5 112	5 088	70 100	273	2 684	3%	3%
57	100	10 000	0,5	7	7	**0,8**	**0,4**	0,0	5 088	5 112	70 100	278	2 627	3%	3%
58	100	10 000	0,5	7	7	**0,8**	**0,2**	0,0	5 130	5 070	70 100	285	2 565	3%	3%
59	100	10 000	0,5	7	7	**0,8**	**0,0**	0,0	5 113	5 087	70 100	275	2 486	3%	3%
60	100	10 000	0,5	7	7	**0,6**	**1,0**	0,0	5 086	5 114	70 100	240	2 905	4%	4%
61	100	10 000	0,5	7	7	**0,6**	**0,8**	0,0	5 170	5 030	70 100	243	2 680	3%	3%
62	100	10 000	0,5	7	7	**0,6**	**0,6**	0,0	5 127	5 073	70 100	243	2 537	3%	3%
63	100	10 000	0,5	7	7	**0,6**	**0,4**	0,0	5 081	5 119	70 100	248	2 492	3%	3%
64	100	10 000	0,5	7	7	**0,6**	**0,2**	0,0	5 085	5 115	70 100	253	2 434	3%	3%
65	100	10 000	0,5	7	7	**0,6**	**0,0**	0,0	5 117	5 083	70 100	248	2 370	3%	3%
66	100	10 000	0,5	7	7	**0,4**	**1,0**	0,0	5 145	5 055	70 100	223	2 890	4%	4%
67	100	10 000	0,5	7	7	**0,4**	**0,8**	0,0	5 098	5 102	70 100	233	2 580	3%	3%
68	100	10 000	0,5	7	7	**0,4**	**0,6**	0,0	5 009	5 191	70 100	226	2 527	3%	3%
69	100	10 000	0,5	7	7	**0,4**	**0,4**	0,0	5 176	5 024	70 100	231	2 355	3%	3%
70	100	10 000	0,5	7	7	**0,4**	**0,2**	0,0	5 052	5 148	70 100	235	2 363	3%	3%
71	100	10 000	0,5	7	7	**0,4**	**0,0**	0,0	5 042	5 158	70 100	227	2 309	3%	3%
72	100	10 000	0,5	7	7	**0,2**	**1,0**	0,0	5 030	5 170	70 100	210	2 868	4%	4%
73	100	10 000	0,5	7	7	**0,2**	**0,8**	0,0	5 063	5 137	70 100	215	2 575	3%	3%
74	100	10 000	0,5	7	7	**0,2**	**0,6**	0,0	5 156	5 044	70 100	221	2 388	3%	3%
75	100	10 000	0,5	7	7	**0,2**	**0,4**	0,0	5 158	5 042	70 100	222	2 305	3%	3%
76	100	10 000	0,5	7	7	**0,2**	**0,2**	0,0	5 084	5 116	70 100	220	2 272	3%	3%
77	100	10 000	0,5	7	7	**0,2**	**0,0**	0,0	5 039	5 161	70 100	215	2 251	3%	3%
78	100	10 000	0,5	7	7	**0,0**	**1,0**	0,0	5 054	5 146	70 100	206	2 755	3%	3%
79	100	10 000	0,5	7	7	**0,0**	**0,8**	0,0	5 102	5 098	70 100	207	2 529	3%	3%
80	100	10 000	0,5	7	7	**0,0**	**0,6**	0,0	5 044	5 156	70 100	210	2 401	3%	3%
81	100	10 000	0,5	7	7	**0,0**	**0,4**	0,0	4 993	5 207	70 100	211	2 331	3%	3%
82	100	10 000	0,5	7	7	**0,0**	**0,2**	0,0	5 145	5 055	70 100	211	2 201	3%	3%
83	100	10 000	0,5	7	7	**0,0**	**0,0**	0,0	5 188	5 012	70 100	214	2 150	2%	3%

Fig. 15. Random graphs generated to test the performance of recommender systems (part 2). Graph generator's parameters and BLCC are defined in Sec 3.1. Neighbors stands for an average number of potentially similar users, 2^{nd} neighbors stands for an average number of items of potentially simialar users.

[20] M. Newman, S. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications," vol. 64, July 2001.
[21] T. Dunning, "Accurate methods for the statistics of surprise and coincidence," *Computational Linguistics*, vol. 19, no. 1, pp. 61–74, 1993.